Workgroup: Independent Submission
Internet-Draft:
draft-ideskog-assisted-token-05
Published: 9 March 2021
Intended Status: Informational
Expires: 10 September 2021
Authors: J. Ideskog   T. Spencer
Curity AB   Curity AB

OAuth 2.0 Assisted Token

Abstract

This document extends the OAuth 2.0 framework to include an additional authorization flow for single page applications called the assisted token flow. It enables OAuth clients written in scripting languages, like JavaScript, to request user authorization using a simplified method compared to other flows. Communication does not rely on redirection of the user agent, but instead leverages HTML's iframe element, child windows, and the postMessage interface. This communication is done using an additional endpoint, the assisted token endpoint.

To contribute to this draft, please feel free to create a pull request from the original source available at https://github.com/curityio/rfc or email the authors.

This note is to be removed before publishing as an RFC.

Status of This Memo

This Internet-Draft will expire on 10 September 2021.

Copyright Notice

Table of Contents

1. Introduction

The OAuth 2.0 protocol flow for Single-page Applications (SPA) defined in this memo, often referred to as the assisted token flow, provides clients written in scripting languages, like JavaScript, with a simplified integration (compared to the implicit or authorization code flow) and ensures that end users are not redirected away from the current page in order to obtain authorization from the resource owner. The communication between the client and the authorization server takes place within an HTML iframe element or child window that is only displayed when interactive user interaction is required; this is the case when authentication and/or authorization are necessary. To communicate the result from this iframe or child window to the client application, HTML's postMessage interface is used instead of the redirection endpoint defined in Section 3.1.2 of [RFC6749]. This difference is important for many SPAs which take time to reload and may not be able to recreate the same state prior to the user being redirected to the authorization server.

Another goal of the assisted token flow is to simplify integration of the client with the authorization server. Though [RFC6749] resulted in a much simpler integration for client applications compared to its predecessor, [RFC5849], developers still struggle with the many inputs required to perform the various OAuth flows. For this reason, the assisted token flow introduces a new endpoint called the assisted token endpoint rather than extending and reusing the token endpoint defined in Section 3.2 of [RFC6749]. As a result, client developers do not need to specify a response_type parameter in the authorization request. This coupled with the use of HTML's postMessage interface for communication between the client and the authorization server means that the redirect_uri and state parameters are also unnecessary. Consequently, client developers only need to provide a client_id, create a dynamic iframe or open a child window, and handle the postMessage that is fired by the authorization server in order to implement OAuth.

This interaction is shown in Figure 1.

```
+----------------------------+                       +---------------+
|           Client           |                       | Authorization |
| +--------------------------+                       |     Server    |
| |              +-----------+                       |               |
| |--(A)-------->|           |--(B)-- Client ------>|               |
| |              |           |       Identifier    |               |
| |              |  Hidden   |                     |               |
| |              |  iframe   |<-(C)-- HTML with ----|               |
| |<-(D)- Token -|           |       postMessage   |               |
| |       or     |           |       including     |               |
| |       error  +-----------+       access token  |               |
| +--------------------------+       or error      |               |
|                            |                     |               |
+----------------------------+                     +---------------+
```

Figure 1: Assisted Token Flow

The assisted token flow illustrated in Figure 1 includes the following steps:

(A) The client creates a hidden iframe element using a scripting language like JavaScript. The src attribute of this iframe is the URL of the assisted token endpoint of the authorization server.

(B) The query string of the src attribute on the iframe includes, at a minimum, the client identifier.

(C) If the user is already logged in and has granted consent to the client, the authorization server immediately returns an HTML document that includes a script that is executing; this fires an event which is communicated to the client using an HTML window.postMessage.

(D) The client handles this event -- the payload of which is either an access token or an error; the client then closes the dynamic iframe without revealing it to the user.

If the resource owner has not authenticated or has not authorized the client, then interaction between the resource owner and the authorization server is required to obtain these. In such a case, the HTML in step (C) of Figure 1 will include an error indicating that user involvement is required. This will be handled by the client in step (D) and login and/or consent will commence. This process is illustrated in Figure 2.

```
+----------------------------+                      +---------------+
|           Client           |                      | Authorization |
| +--------------------------+                      |     Server    |
| |              +-----------+        Client        |               |
| |--(E)-------->|           |--(F)-- Identifier -->|               |
| |              |  Visible  |                      |               |
| |              |  iframe   |--(G)-- User -------->|               |
| |              |  or child |        interaction   |               |
| |<-(I)- Token -|  window   |                      |               |
| |              |           |<-(H)-- HTML with ----|               |
| |              +-----------+        postMessage   |               |
| +--------------------------+        including     |               |
|                            |        access token  |               |
+----------------------------+                      +---------------+
```

Figure 2: Assisted Token Login and/or Consent Flow

The flow shown in Figure 2 includes the following steps:

(E)
The client creates a visible iframe or pops open a child window after receiving an indication from the authorization server that user interaction is required. As in the previous flow, the src attribute value of this iframe or the input to the open method of the user agents's window object is the URL of the authorization server's assisted token endpoint.

(F) The query string of this URL includes, at a minimum, the client identifier.

(G) The authorization server prompts the resource owner to authenticate and/or authorize the client.

(H) The authorization server returns an HTML document that includes a script that is executed; this fires an event which is communicated to the client using an HTML window.postMessage.

(I) The client handles this event -- the payload of which is an access token; the client then closes the iframe or child window that it previously opened to facilitate user interaction.

2. Terminology

The key words "**MUST**", "**MUST NOT**", "**REQUIRED**", "**SHALL**", "**SHALL NOT**", "**SHOULD**", "**SHOULD NOT**", "**RECOMMENDED**", "**NOT RECOMMENDED**", "**MAY**", and "**OPTIONAL**" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

Handle Token
An opaque token that refers to internal data of the authorization server (e.g., the user ID, scope of the token, etc.) as described in Section 3.1 of [RFC6819].

All other terms used in this document are as defined in [RFC6749]. Unless otherwise noted, all the protocol parameter names and values are case sensitive.

3. Assisted Token Endpoint

The means through which the client obtains the location of the assisted token endpoint is either by using the authorization server's metadata as set forth in Section 6, the service documentation, or some other method that is beyond the scope of this specification.

The endpoint URI **MAY** include an application/x-www-form-urlencoded formatted (per Appendix B of [RFC6749]) query component (see Section 3.4 of [RFC3986]), which **MUST** be retained when adding additional query parameters. The endpoint URI **MUST NOT** include a fragment component.

Since requests to the assisted token endpoint result in the transmission of clear-text credentials (in the HTTP request and response), the authorization server **MUST** require the use of TLS as described in Section 1.6 of [RFC6749] when sending requests to the assisted token endpoint.

The client **MUST** use the HTTP GET method when making access token requests to this endpoint.

Parameters sent without a value **MAY** be treated as if they were omitted from the request. The authorization server **MUST** ignore unrecognized request parameters. Request and response parameters **MUST NOT** be included more than once.

After completing its interaction with the resource owner, the authorization server will fire an event using the HTML postMessage interface. This message **MUST NOT** be posted to all origins, denoted by *. Instead, the authorization server **MUST** post this message only to the client's allowed origin(s) previously established with the authorization server during the client registration process.

3.1. Access Token Scope

Like the authorization and token endpoints, the assisted token endpoint allows the client to specify the scope of the access request using the scope request parameter. In turn, the authorization server uses the scope response parameter to inform the client of the scope of the access token issued. Unlike the typical behavior of those endpoints, however, access tokens issued by the authorization server using the assisted token endpoint **MUST** use the client's pre-configured scope or the authorization server's pre-defined default if none have been configured for the client.

If the client did not include a scope request parameter or if the issued access token scope is different from the one requested by the client, the authorization server **MUST** include the scope response parameter to inform the client of the actual scope granted. Even when the scope of the issued access token is the same as the one requested by the client, the authorization server **SHOULD** include the scope response parameter.

The format, constraints, and grammar of the scope parameter value is as defined in Section 3.3 of [RFC6749].

The authorization server **SHOULD NOT** return an error if a scope has not been pre-configured for the client; only if the authorization server does not have a pre-defined default scope.

3.2. Cross-Origin Support

The assisted token endpoint **MAY** support CORS as defined in [W3C.WD-cors-20120403].

4. Protocol

4.1. Assisted Token Request

The assisted token request is an HTTP GET request constructed by the client with the following parameters provided on the query string:

client_id
REQUIRED. The client identifier as described in Section 2.2 of [RFC6749].

for_origin
OPTIONAL. The origin of the client in case multiple allowed origins are configured with the authorization server and support for user agents that do not support [CSP-2] but only X-Frame-Options (as defined in [RFC7034]). See Section 8.1 for details.

prompt
OPTIONAL. Space delimited, case sensitive list of [ASCII] string values that can be used to determine the login state of the resource owner at the authorization server. The defined values are:

none
The authorization server **MUST NOT** display any authentication or consent user interface pages. An error is returned if the user is not already authenticated or if the client has not received consent (either explicitly by the resource owner or by the authorization server's configuration of the client) or if the authorization server cannot fulfill other conditions for processing. This can be used as a method to probing for existing authentication and/or consent.

consent
The authorization server **SHOULD** prompt the user for consent before returning information to the client. If it cannot obtain consent, it **MUST** return an error.

Other values may be provided in this list; the authorization server **MUST** ignore them without producing an error if it cannot understand them.

scope

OPTIONAL. The scope of the access request as described in Section 3.1.

4.2. Assisted Token Response

The response from the assisted token endpoint is an HTML document that executes a script which invokes the HTML postMessage interface to send a message to either the parent window (in the case shown in Figure 1) or the opener. The origin that this event is posted to is that of the client. The contents of this message is a JSON object with the following fields:

access_token

REQUIRED. The access token issued by the authorization server.

expires_in

RECOMMENDED. The lifetime in seconds of the access token.

scope

REQUIRED. The scope of the access token as described in Section 3.1.

sub

RECOMMENDED. A locally unique and never reassigned identifier within the authorization server for the user, which is intended to be consumed by the client. The sub value is a case sensitive string.

token_type

REQUIRED. The type of the token issued as described in Section 7.1 of [RFC6749]. The value is case insensitive.

This JSON object **MAY** contain additional fields. If the client does not understand or recognize such additional fields, it **MUST** ignore them.

4.3. Error Response

As with a successful response, an error is returned to the client using an the HTML postMessage interface. Such an error is returned whenever the resource owner denies the access request or whenever the request fails for reasons other than the origin of the client being disallowed to frame the assisted token endpoint. The error message includes a JSON object with the following fields:

error

REQUIRED. A single [ASCII] error code from the following:

invalid_request

The request is missing a required parameter, includes an invalid parameter value, includes a parameter more than once, or is otherwise malformed.

unauthorized_client
The client is not authorized to request an access token using this method.

access_denied
The resource owner or authorization server denied the request.

consent_required
The client includes a prompt value of consent, but consent by the resource owner was required.

interaction_required
The client included a prompt value of none, but either the user needed to authenticate or consent to the client's access or some other requirement of the authorization server prevented it from providing access without some form of user interaction.

unsupported_response_type
The authorization server does not support obtaining an access token using this method.

invalid_scope
The requested scope is invalid, unknown, or malformed.

server_error
The authorization server encountered an unexpected condition that prevented it from fulfilling the request. (This error code is needed because a 500 Internal Server Error HTTP status code cannot be returned to the client directly.)

temporarily_unavailable
The authorization server is currently unable to handle the request due to a temporary overloading or maintenance of the server. (This error code is needed because a 503 Service Unavailable HTTP status code cannot be returned to the client directly.)

Values for the error parameter **MUST NOT** include characters outside the set %x20-21 / %x23-5B / %x5D-7E.

error_description
OPTIONAL. Human-readable [ASCII] text providing additional information, used to assist the client developer in understanding the error that occurred. Values for the error_description

parameter **MUST NOT** include characters outside the set %x20-21 / %x23-5B / %x5D-7E.

error_uri

OPTIONAL. A URI identifying a human-readable web page with information about the error, used to provide the client developer with additional information about the error. Values for the error_uri parameter **MUST** conform to the URI-reference syntax and thus **MUST NOT** include characters outside the set %x21 / %x23-5B / %x5D-7E.

5. Client Metadata

The authorization server **MAY** allow dynamic clients to request the use of the assisted token flow when registering. Such a client may indicate that it will interact with the authorization server using the assisted token flow by including the string element urn:ietf:params:oauth:grant-type:assisted_token in the array associated with the grant_types metadata field sent to the client registration endpoint (as defined in Section 3 of [RFC7591]). If the authorization server allows the client to register with this grant type, the grant_types included in the response **MUST** include the value urn:ietf:params:oauth:grant-type:assisted_token. The inclusion of this value in the grant_types field is done despite the fact that the client will not use this grant type at the token endpoint but rather the assisted token endpoint (see Section 3).

If dynamic clients are allowed to register themselves with the authorization server, then additional verification of the origins include in the request **MUST** be performed in some manner. The way in which this is accomplished by the authorization server is not defined by this specification. Because this additional verification is required and because clients using the assisted token flow are typically ephemeral in nature, it is **NOT RECOMMENDED** that the authorization server support dynamic client registration of the urn:ietf:params:oauth:grant-type:assisted_token grant type.

The following client metadata field is defined by this specification. It **MAY** be included in a registration request, as set forth in Section 2 of [RFC7591].

allowed_origins

Array of origin strings for use in sending messages from the authorization server to the client using the HTML's postMessage interface.

6. Authorization Server Metadata

Support for the assisted token flow **SHOULD** be declared in the authorization server's metadata (as defined in [RFC8414]) with the following metadata:

assisted_token_endpoint
RECOMMENDED. URL of the authorization server's assisted token endpoint defined in Section 3.

grant_types_supported
RECOMMENDED. A JSON array specified in Section 2 of [RFC8414] which **SHOULD** contain the value urn:ietf:params:oauth:grant-type:assisted_token as defined in Section 5.

7. IANA Considerations

7.1. OAuth URI Registration

This specification registers the following values in the IANA "OAuth URI" registry [IANA.OAuth.Parameters] established by [RFC6755].

7.1.1. Registry Contents

*URN: urn:ietf:params:oauth:grant-type:assisted_token

*Common Name: Assisted token flow grant type for OAuth 2.0

*Change controller: CURITY: info@curity.io

*Specification Document: Section 5 of [[this specification]]

7.2. OAuth Parameters

This specification registers the following values in the IANA "OAuth Parameter" registry [IANA.OAuth.Parameters] established by [RFC6749].

7.2.1. Registry Contents

*for_origin

*Parameter usage location: authorization request

*Change controller: CURITY: info@curity.io

*Specification Document: Section 4 of [[this specification]]

7.3. OAuth 2.0 Authorization Server Metadata

This specification registers the following values in the IANA "OAuth 2.0 Authorization Server Metadata" registry [IANA.OAuth.Parameters] established by [RFC8414].

7.3.1. Registry Contents

*Metadata name: assisted_token_endpoint

*Metadata Description: The Assisted Token Endpoint.

*Change controller: CURITY: info@curity.io

*Specification Document: Section 6 of [[this specification]]

8. Security Considerations

In addition to all the security considerations discussed in [RFC6819], the following security considerations **SHOULD** be taken into account.

8.1. Framing

Due to the use of an iframe to host the assisted token endpoint, the authorization server **MUST** take precautions to ensure that only trusted origins are allowed to frame it. The authorization server **MUST** prevent any origin from framing the assisted token endpoint except ones deemed trustworthy. This attestation may come byway of explicit administrative action or automated techniques available to the authorization server. Additional care is **REQUIRED** if dynamic client registration is supported or clients may be registered in a self-service manner.

One such mechanism that **MAY** be deployed is Content Security Policy [CSP-2]. This protocol **SHOULD** be used on the assisted token endpoint (and, if applicable, other endpoints used to authenticated the user in a specific deployment) to prevent framing from unauthorized origins. Using CSP allows the authorization server to specify multiple origins in a single response header field and to constrain these using flexible patterns (see [CSP-2] for details). This standard provides a robust mechanism for protecting against click-jacking by using policies that restrict the origin of frames (using frame-ancestors) together with those that restrict the sources of scripts allowed to execute on an HTML page (by using script-src). A non-normative example of such a policy is shown in the following listing:

```
HTTP/1.1 200 OK
Content-Security-Policy: frame-ancestors https://a.example.org:8000
Content-Security-Policy: script-src 'self'
X-Frame-Options: ALLOW-FROM https://a.example.org:8000
...
```

Figure 3: Example CSP that will help protect against click-jacking

Because some user agents do not support [CSP-2], this technique **SHOULD** be combined with others. In particular, the authorization server **SHOULD** return an X-Frame-Options response header on the assisted token endpoint (and, if applicable, other endpoints used to authenticate the user and authorize the client in a specific deployment). As defined in [RFC7034], this header will cause user agents that support it (but not CSP) to block framing from any origin that is not specified in this header's value. Because this header's value can only include one origin, the framer is **RECOMMENDED** to use the for_origin parameter (as specified in Section 4.1) with its own origin as the value.

Some user agents do not support [RFC7034] nor [CSP-2]. Therefore, the authorization server **SHOULD** include a frame-busting script like that shown in Figure 7 of [FRAME-BUSTING]. Such a script would use JavaScript to break out of any unauthorized origin that is framing the assisted token endpoint. The authorization server **MAY** simply break out of all frames in case [RFC7034] and [CSP-2] are unsupported by the user agent, though this would render the assisted token flow non-functional. The choice of whether or not this drastic countermeasure should be employed depends on the user agents being targeted in a certain deployment.

When the authorization server and client are provided by separate parties, it is important that the resource owner is able to distinguish the two. One safe way of doing so is by examination of the user agent's address bar by the resource owner where the validity of the certificate and location can be verified. In such situations, whenever user interaction is required, the client **MUST** open the assisted token endpoint in a new browser window rather than a hidden iframe. The authorization server **MUST** take measures deemed appropriate in a deployment to ensure that the client has not framed the user's manual interaction; however, the necessity for interactive user authentication and/or consent **SHOULD** be possible for the client to determine in a hidden iframe.

8.2. WebAuthn

[WebAuthn] **MAY** be used to ensure that the user is authenticated to the authorization server in a safe manner. This protocol offers authentication processes that are resistant to man-in-the-middle attacks. The API defined by this standard can be used to verify that user authentication is performed at the expected origin. This is superior to manual verification of the origin in the address bar of the browser by the resource owner. It also provides a mechanism to restrict the client framing the assisted token endpoint to be in the same origin. When it is not, the use of [WebAuthn] will force the client to use a separate window for user authentication.

8.3. Handle Tokens

Because the client applications that use the assisted token flow are written in scripting languages like JavaScript and are hosted in Web pages, users may keep such applications open in their user agents for a prolonged period of time. During such period, the token issued to the client may expire or be revoked. To ensure that such expired tokens left remnant in the user agent are benign, a Handle Token **SHOULD** always be issued by the assisted token endpoint. This ensures that no identity data is exposed (even when the token is not yet expired) and that a revoked token does not increase risks.

8.4. Warning Against Untrusted Scripts

As admitted in Section 8 of [RFC6454], preventing exfiltration of cookies, tokens, and other such credentials in web browsers has historically proven difficult to implement. Instead, the same-origin policy has emerged as the cornerstone of security for such user agents. Using this security model, it is not possible to prevent access to a token issued to a client if that client includes nefarious scripts from untrustworthy sources that have access to the Document Object Model (DOM) where the token is stored. For this reason, the authorization server **MUST** warn client application developers who interact with the assisted token endpoint *not* to use untrusted scripts in their applications. This warning **SHOULD** at least be conveyed through the documentation but **MAY** also be provided through other mechanisms.

8.5. Origin of Event and Authorization Server

As described in Section 4.1, the authorization server will return an access token to the client using HTML's postMessage interface. The receiver of this message is provided with an event object that contains an origin property. A client application **MUST** compare this with that of the authorization server before consuming the message. Otherwise, it runs the risk of processing messages posted from untrusted origins. An example of a proper message handler is shown in the following non-normative, JavaScript listing:

```
<script type="text/javascript">
    window.addEventListener("message", function(evt) {
        if (evt.origin !== "https://oauth-server.example.com")
            return; // Ignore event from untrusted source

        ...
    });
</script>
```

Figure 4: Comparing the origin of the postMessage event with that of the authorization server

8.6. Token Storage

Most client applications that use the assisted token flow will maintain the access token issued by the authorization server in a persisted state; this will commonly be an HTTP cookie or local storage. This is necessary, for instance, to create a pleasing user experience when a user navigates away from the application in their web browser and then returns. To ensure that the token is stored safely, the authorization server **MAY** provide application developers with guidance in the accompanying documentation on how to safely persist tokens. The authorization server **MAY** also provide script libraries that perform this action according to best common practices. The authorization server **MAY** also store the token in an HTTP cookie in its own DNS domain (rather than that of the client) using the assisted token endpoint's path. In some cases, this would elevate any storage requirements from the client application developer. Besides simplifying the programming model for developers, this technique allows the authorization server to check the validity of the token and determine if the token has expired or if the associated grant has been revoked in subsequent requests to the assisted token endpoint; this will be possible because the requests might include the token in the HTTP Cookie request header. In such cases, the authorization server can take the appropriate action, such as authenticating the user anew or request consent, before issuing a new token. If the token is stored by the application, however, this kind of verification cannot be performed by the authorization server without an explicit request to validate a stored token.

9. Privacy Considerations

In some deployments, the assisted token endpoint may be served from a distinct domain from that of the client. In such cases, the client will be a third-party domain, and the resource owner's user agent may prevent the authorization server from storing any third-party cookies. If the authorization server requires state to be persisted when performing the assisted token flow, it **SHOULD** provide a privacy-preserving mechanism to store and retrieve its state even when the assisted token endpoint is hosted on a distinct domain from that of the client. The technical details of how to accomplish this are implementation specific, and are beyond the scope of this specification. If the authorization server does not support clients that are hosted from a third-party domain, it **MUST** indicate this to the client through some mechanism (e.g., its associated documentation).

10. Normative References

[ASCII] American National Standards Institute, "Coded Character Set -- 7-bit American Standard Code for Information Interchange", ANSI X3.4, 1986.

[IANA.OAuth.Parameters] IANA, "OAuth Parameters", <http://www.iana.org/assignments/oauth-parameters>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <https://www.rfc-editor.org/info/rfc6749>.

[RFC7591] Richer, J., Ed., Jones, M., Bradley, J., Machulak, M., and P. Hunt, "OAuth 2.0 Dynamic Client Registration Protocol", RFC 7591, DOI 10.17487/RFC7591, July 2015, <https://www.rfc-editor.org/info/rfc7591>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8414] Jones, M., Sakimura, N., and J. Bradley, "OAuth 2.0 Authorization Server Metadata", RFC 8414, DOI 10.17487/RFC8414, June 2018, <https://www.rfc-editor.org/info/rfc8414>.

11. Informative References

[CSP-2]

West, M., Barth, A., and D. Veditz, "Content Security Policy Level 2", July 2015, <https://www.w3.org/TR/CSP2>.

[FRAME-BUSTING] Rydstedt, G., Bursztein, E., Boneh, D., and C. Jackson, "Busting frame busting: a study of clickjacking vulnerabilities at popular sites", July 2010, <http://seclab.stanford.edu/websec/framebusting/framebust.pdf>.

[RFC5849] Hammer-Lahav, E., Ed., "The OAuth 1.0 Protocol", RFC 5849, DOI 10.17487/RFC5849, April 2010, <https://www.rfc-editor.org/info/rfc5849>.

[RFC6454] Barth, A., "The Web Origin Concept", RFC 6454, DOI 10.17487/RFC6454, December 2011, <https://www.rfc-editor.org/info/rfc6454>.

[RFC6755] Campbell, B. and H. Tschofenig, "An IETF URN Sub-Namespace for OAuth", RFC 6755, DOI 10.17487/RFC6755, October 2012, <https://www.rfc-editor.org/info/rfc6755>.

[RFC6819] Lodderstedt, T., Ed., McGloin, M., and P. Hunt, "OAuth 2.0 Threat Model and Security Considerations", RFC 6819, DOI 10.17487/RFC6819, January 2013, <https://www.rfc-editor.org/info/rfc6819>.

[RFC7034] Ross, D. and T. Gondrom, "HTTP Header Field X-Frame-Options", RFC 7034, DOI 10.17487/RFC7034, October 2013, <https://www.rfc-editor.org/info/rfc7034>.

[W3C.WD-cors-20120403] Kesteren, A., "Cross-Origin Resource Sharing", World Wide Web Consortium LastCall WD-cors-20120403, 3 April 2012, <http://www.w3.org/TR/2012/WD-cors-20120403>.

[WebAuthn] Balfanz, D., Czeskis, A., Hodges, J., Jones, J.C., Jones, M., Kumar, A., Liao, A., Lindemann, R., and E. Lundberg, "Web Authentication: An API for accessing Public Key Credentials", World Wide Web Consortium (W3C) Recommendation, 4 March 2019, <https://www.w3.org/TR/2019/REC-webauthn-1-20190304>.

Appendix A. Acknowledgements

The following individuals contributed ideas, feedback, and wording to this specification:

Mark Dobrinic, Karl McGuinness, Renato Athaydes, Daniel Lindau, Louis Jannett

Appendix B. Document History

[[to be removed by the RFC editor before publication as an RFC]]

This section is to be removed before publishing as an RFC.

-05

- Clarified text and figures

- Added note about DCR not being recommended

- Fixed change controller for IANA registry entries

- Clarified and reorganized framing and address bar security considerations based on L. Lannett's feedback

- Added WebAuthn security consideration

-04

- Updated IANA references to use the "short version" without file extension or extra path information.

- Updated repository reference.

-03

- Updated repository links.

-02

- Updated section 8.1 with regards to the CSP headers.

- Updated authorization server metadata in section 6.

- Updated IANA Considerations section 7.

-01

- Updated to v. 3 XML format

-00

- Initial draft.

Authors' Addresses

Jacob Ideskog
Curity AB

Email: jacob@curity.io

Travis Spencer
Curity AB

Email: travis@curity.io